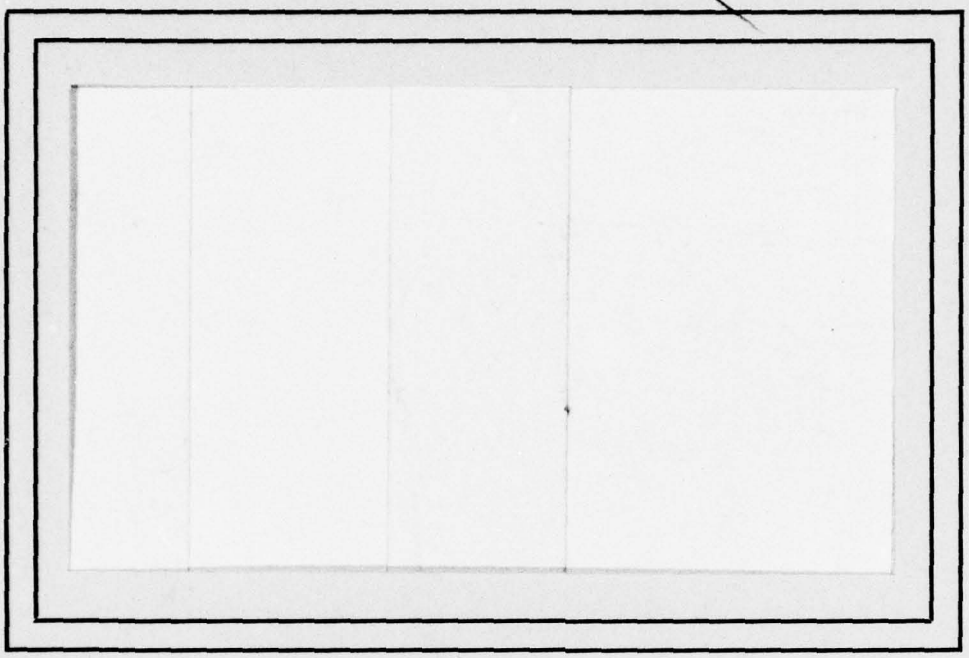
(1) LEVEL II



UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND
20742

(1) LEVEL II

(14) CSC-TR-741 (9) Technical rept., (12) 49

(11) Mar[redacted]79

(15) TR-741
DAAG-53-76C-0138
DARPA Order-3206

(6) REGION REPRESENTATION:
QUADTREES FROM BOUNDARY CODES.

(10) Hanan/Samet
Computer Science Department
University of Maryland
College Park, MD 20742

ABSTRACT

An algorithm is presented for constructing a quadtree for a region given its boundary in the form of a chain code. The algorithm makes use of some geometrical properties of the region to enable the detection of the maximal size blocks of the region without actually visiting all of the subblocks of the maximal size block. Analysis of the algorithm reveals that its execution time is proportional to the product of the perimeter and the log of the diameter of the region.

The support of the Defense Advanced Research Projects Agency and the U.S. Army Night Vision Laboratory under Contract DAAG-53-76C-0138 (DARPA Order 3206) is gratefully acknowledged, as is the help of Kathryn Riley in preparing this paper. The author has also benefited greatly from discussions with Charles R. Dyer and Azriel Rosenfeld.

403 018

1. Introduction

Region representation is an important issue in image processing, cartography, and computer graphics. There are numerous representations currently in use (see [DRS] for a brief review). In this paper we focus our attention on a pair of such representations termed the chain code [Freeman] and the quadtree [KD]. In [DRS] an algorithm is presented for obtaining a chain code from a quadtree. In this paper, we present an algorithm for obtaining the quadtree corresponding to the chain code for a given region. Such algorithms are important because each representation is well suited for a specific set of operations on a region. For example, the chain code is very compact and facilitates the detection of boundary features such as corners and concavities. The quadtree is useful because it enables a hierarchical representation as well as facilitating operations such as region union and intersection.

In the remainder of the paper we briefly review the definitions of a chain code and a quadtree. Sections 2-6 present and analyze our algorithm. Included is an informal description of the algorithm and a rationale for its various steps. The formal presentation of the algorithm is given using ALGOL-like [Naur 60] procedures. Note that our algorithms are different from those given in [Hunter] in the domain of computer graphics where each node stores the list of coordinate points that describe the

polygon from which the quadtree was constructed. We don't make use of any information of this nature.

We assume that a region is a simply-connected subset of a 2^n by 2^n array, without holes, which we view as being composed of unit square "pixels." We further assume that the region is four-connected--i.e., blocks touching only at a corner are not adjacent. The chain code is a representation of the boundary consisting of a sequence of unit vectors in the principal directions (i.e., N,E,S,W). The directions can be represented by numbers, e.g. let i represent the direction 90i° where i=0,1,2,3. As an example, consider the region in Figure 1a. The chain code of its boundary, moving clockwise standing at the leftmost upper corner. is $0^7 3030^2 1030303^6 2^5 32^2 121^2 2^5 1^7$. Note that generalized chain codes involving more than four directions can be used although this is not done here.

The quadtree is an approach to region representation based on a successive subdivision of the array into quadrants. In essence, we continually subdivide the array into quadrants, subquadrants,... until we obtain blocks (possibly single pixels) which are either entirely in the region or entirely disjoint from it. This process is represented by a tree of out-degree 4 in which the root node represents the entire array, the four sons of the root node represent the quadrants, and the terminal nodes correspond to those blocks of the array for which no

further subdivision is necessary. Note that our quadtrees are different from those introduced by [FB] to represent point data in two-dimensional space. For example, Figure 1b is a block decomposition of the region in Figure 1a while Figure 1c is the corresponding quadtree. In general, BLACK square nodes represent nodes that are present in the region while WHITE square nodes represent nodes that are absent in the region. Circular nodes correspond to non-terminal nodes, also referred to as GRAY nodes.

2. Definitions and notation

Let each node in a quadtree be stored as a record containing six fields. The first five fields contain pointers to the node's father and its four sons labeled NW, NE, SE, and SW. Given a node P and a son I, these fields are referenced as FATHER(P) and SON(P,I), respectively. At times it is useful to use the function SONTYPE(P) where SONTYPE(P)=Q iff SON(FATHER(P), Q)=P. The sixth field, named NODETYPE, describes the contents of the block of the region which the node represents--i.e., WHITE, if the block contains no pixels in the region; BLACK, if the block contains only pixels in the region; and GRAY, if it contains pixels of both types. Alternatively, BLACK and WHITE nodes are terminal nodes while GRAY nodes are non-terminal nodes.

Let the four sides of a node's block be called its N,E,S, and W sides. They are also termed its boundaries. The four directions that can be represented by a four direction chain code are labeled E, S, W, and N. A clockwise chain code is being used which means that the region is always to the right of the boundary represented by the code. BOUNDARY (D) indicates the type of boundary represented by a code in direction D; i.e., directions E,S,W, and N correspond to northern, eastern, southern, and western boundaries respectively. OPDIR(D) is a direction 180° in opposition to direction D. The function COMPARE(D1,D2)

indicates the relationship of direction D2 to direction D1. It has values of SAME, CLOCKWISE, COUNTERCLOCKWISE, and OPPOSITE. For example, COMPARE(E,S) is CLOCKWISE. Figure 2a shows the relationship between quadrants of a node and directions of a chain code, and Figure 2b shows the relationship between quadrants of a node and its boundaries.

There are a number of other functions which are useful in describing the relationships between nodes, quadrants, and boundaries. In the following definitions we shall use P, Q, and R as node variables, I and J as quadrant variables, and B and C as boundary variables. QUAD(B,C) is defined to be the quadrant which is bounded by boundaries B and C (if B and C are opposite boundaries, then the function is undefined); e.g., QUAD(N,W)=NW. Quadrants are ordered NW, NE, SE, and SW in the clockwise direction. CQUAD(I) returns I's neighbor in the clockwise direction; e.g. CQUAD(SW)=NW. Similarly, CCQUAD(I) returns I's neighbor in the counterclockwise direction; e.g., CCQUAD(SW)=SE. The predicate ADJ(B,I) is true if and only if quadrant I is adjacent to boundary B of the node's block; e.g., ADJ(N,NE)=true. The function REFLECT(B,I) yields the quadrant which is adjacent to I along boundary B of the block represented by qudrant I; e.g., REFLECT(W,NW)=NE, REFLECT(E,NW)=NE, REFLECT(N,NW)=SW, and REFLECT(S,NW)=SW. COMMONBOUNDARY(I,J) denotes the boundary that non-WHITE (i.e., BLACK or GRAY)

quadrant J shares with its adjacent WHITE quadrant I. Note that I and J are quadrants of the same node. For example, COMMONBOUNDARY(NW,NE)=W.

Given a quadtree corresponding to a region represented by a 2^n by 2^n array, we say that the root node is at level n, and that a node at level i is at a distance of n-i from the root of the tree. In other words, for a node at level i, we must ascend n-i FATHER links to reach the root of the tree. Note that the furthest node from the root of the tree is at level Ø.

It is also useful to further characterize BLACK nodes in terms of their relationship to the boundary represented by the chain code. An encoding is used that stipulates for each BLACK node the part of the boundary, if any, that it represents. Ø indicates that the node does not have any side that is entirely on the boundary (referred to as NOBORDERBLACK). 1, 2, 4, and 8 correspond to sides that are entirely on northern, eastern, southern, and western boundaries, respectively. Note that these types are additive. For example, a node whose sides serve as southern and western boundaries has type 12. For completeness, let GRAY and WHITE be represented by 16 and 17, respectively. The function CODE(B) yields the code corresponding to a boundary of type B (B is N,E,S,W). The predicate BLACK(P) indicates if NODETYPE(P) is BLACK.

3. Informal description of the algorithm

The quadtree construction algorithm has two phases. The first phase traces the boundary in the clockwise direction and constructs a tree. During this process all nodes appearing on the boundary (i.e., adjacent to it and within the region being traced) are colored BLACK. All non-terminal nodes are colored GRAY. Once the border has been traced, some of the non-terminal nodes may have sons that were not visited; yet, they are nevertheless part of the region. After the first phase of the algorithm, these sons are represented by NULL links. The second phase of the algorithm determines whether such NULL links correspond to BLACK or WHITE nodes, and colors them appropriately. In addition, whenever a GRAY node is found (i.e., a non-terminal node) whose four sons are BLACK, the sons are removed and the GRAY node becomes a BLACK node. In the following we discuss the two phases of the algorithm in greater detail.

Phase One

Choose any link in the chain code at random and create a node for it, say P. Now, examine the next link in the chain code, say NEW, and compare its direction with that of the immediately preceding link, say OLD. There are three possibilities: (1) if the code maintains its direction, then we may need to add a node, say Q, which is a neighbor of P in the OLD direction (see Figure 3a). (2) If the code turns clockwise, then there is no

need to add a node (see Figure 3b). (3) If the code turns counterclockwise, then we may have to add two nodes. First, a node, say Q, is added that is a neighbor of P in direction OLD. It is said to abut the border. Second, a node, say R, is added that is a neighbor of Q in direction NEW (see Figure 3c).

Note that a BLACK node is added to the quadtree the first time it is encountered to the right of a link in the chain code or when it abuts a link which has turned counterclockwise (e.g., node Q in Figure 3c). However, since a node can appear to the right of more than one link, it is clear that it can be encountered more than once when traversing the boundary. Nevertheless, each time a node is encountered to the right of a link its NODETYPE field is marked appropriately to reflect the fact that it adjoins the boundary. This information is used in the second phase of the algorithm.

As an example of the first phase of the algorithm consider the region represented by the chain code in Figure 4. We have labeled the links from a to j and the nodes within the region from A to E. Figure 5a-5j shows how the corresponding quadtree is constructed where Figure 5α shows the quadtree after traversing link α. Figure 5d has been decomposed into two components 5d(i) and 5d(ii) in order to reflect the counterclockwise turn between links c and d. We also indicate in Figure 5 how the value of NODETYPE is constructed along with the tree. Note that the resulting quadtree in Figure 5j is not optimal in the sense

that nodes E, A, B, C should be merged (see Figure 6). This is achieved by the second phase of the algorithm.

During the process of constructing the quadtree we see that nodes may have to be created. This is done whenever we attempt to find a neighbor and fail. More formally, finding a neighbor of a node along a designated boundary consists of traversing FATHER links until a common ancestor is found. Once the ancestor is found, we descend along a path that is reflected about the axis formed by the common boundary. For example, in determining the western neighbor of node B in Figure 4 (i.e., the transition from Figure 5c to 5d(i)), we ascend the FATHER link of B (i.e., up the SE link) and then descend to the node which will contain C (i.e., reflect about the W axis which causes us to descend the SW link). As a more complex example, finding the southern neighbor of node C in Figure 4 (i.e., the transition from Figure 5d(i) to 5d(ii)) requires ascending two FATHER links from node C (i.e., the SW and NE links). Notice that this time there was no common ancestor and thus one was created (recall that this situation also arose when we attempted to find the southern neighbor of node A in Figure 4 and the related transition from Figure 5a to 5b). At this point, we once again descend along a path reflected about the S axis (i.e., the SE and NW links in order) and add GRAY nodes as necessary. The terminal node is colored BLACK.

The situation arising when there is no common ancestor merits further discussion. For example, suppose the neighbor of P in direction S is desired and traversing P's FATHER links in search of a common ancestor leads to node Q with FATHER(Q) being NULL. In this case a node is created, say R, which serves as the FATHER of Q. The question remains as to the quadrant of R that is to contain Q. We choose a quadrant, I, such that R is a common ancestor of both P and P's neighbor in direction S. If this were not done, then the search for a common ancestor would have to continue. Note that we actually have a choice for the value of I, i.e., QUAD(OPDIR(S),BOUNDARY(S)) and QUAD(OPDIR(S),BOUNDARY(OPDIR(S))). From Figure 5a to 5b, node A's neighbor in the southerly direction is sought. Node A's FATHER link is NULL and thus we add a node having A as its NE son.

As a final comment observe that the quadtree resulting from phase one only contains GRAY (non-terminal) and BLACK nodes. Furthermore, the BLACK nodes must be adjacent to the border or abut on it. There are no WHITE nodes in the tree or nodes that are interior to the region that do not touch or abut the border. These nodes are represented by the NULL links of the GRAY nodes. Figures 7 and 8 show the result of the application of phase one to the region represented by Figure 1a. Figure 7 serves to correlate Figure 8 with Figure 1. The blocks in the figures have

been labeled ccording to the order in which they were visited--e.g., block 1 was visited prior to block 2, etc.

Phase Two

This phase has two goals. The first is to fill in all NULL links of non-terminal nodes. The second is to convert GRAY nodes that have four BLACK sons to BLACK nodes. This process is achieved by moving from the border of the region towards the center.

Prior to describing the algorithm in detail we make the following observation. An absent son to eventually correspond to a BLACK node must be totally surrounded by BLACK nodes since otherwise it would have been on the border and not absent. Therefore, if any absent son is BLACK, say P, then all remaining absent sons must also be BLACK since by the preceding statement they must have at least a partial BLACK component in order to be able to surround P. Similarly, if an absent son is WHITE, then all remaining absent sons must also be WHITE. For example, see Figure 1c where the presence of BLACK node K_1 implies the presence of BLACK nodes K_2 and K_3. Similarly for the absence of the WHITE nodes surrounding BLACK node U.

Traverse the tree in a clockwise order (say NW, NE, SE, SW). For each GRAY node encountered, say P, if any of the sons is absent (i.e., a NULL link), then choose one such son, say R, that is adjacent to a GRAY or BLACK son, say Q. Consider the nodes

in the subtree rooted at Q that are adjacent to the boundary shared by R and Q. If any of these nodes are WHITE or are BLACK with a border along the shared boundary, then R and the remaining absent sons of P are WHITE nodes. Otherwise, R and the remaining absent sons are BLACK nodes. If all sons are BLACK, then the GRAY node is replaced by a black node and the sons are deleted. Prior to deleting the four sons, NODETYPE(P) must be set. Its value is obtained by traversing the four sons in clockwise order. Whenever two adjacent sons both have borders along the traversing direction, then node P also has a border along this direction. For example, in Figure 5j nodes E and A have northern boundaries, A and B have eastern boundaries, and C and E have western boundaries. Figure 6 contains the resulting tree where the merged node has a NODETYPE value of 14.

Note that NODETYPE only indicates whether the entire boundary of a node along a given direction is on the border. However, no information is lost in this case (i.e., information with respect to part of a side being on the border) because such a node, say P, will never be adjacent to an absent node along this boundary and thus the boundary code of P in this direction will never be examined again. Recall that the only time the second phase of the algorithm revisits a node is when it is adjacent to the border of an absent node. For example, in Figure 4, the NODETYPE of the node resulting from the merger of nodes A, B, C, and E does not have a component representing a southern border.

However, by virtue of the presence of the adjacent node D it can never have an absent southern neighbor.

Figures 1b and 1c show the result of the application of phase two to the region represented by Figure 1a and the quadtree of Figure 8. Figure 1b enables the correlation of Figure 1c with Figure 1a. The blocks in Figure 1b have been labeled according to the order in which they were visited--i.e., block A prior to block B, block Z prior to block AA, block AA prior to block BB, etc. Note that blocks K_1, K_2, and K_3 have been labeled with the same letter in order to indicate that only block K_1 was visited. Since K_1 was a NULL link in Figure 7, having been determined to be BLACK, blocks K_2 and K_3 are also BLACK and hence need not be visited.

The observant reader will have seen that during phase one no WHITE Nodes were added to the tree. Clearly, this need not be so. However, if WHITE nodes were to be added, then the merging process performed by FILLIN would have to be applied to them as well as to BLACK nodes. For example, in Figure 1a, the pair of nodes adjacent to nodes 9, 11, 12, and 14 are known to be WHITE. However, by failing to mark them as such, we need not merge them with their northern neighbors at a subsequent step of phase two. In other words, by treating WHITE nodes as absent sons in phase one, phase two is only confronted with maximal sized WHITE nodes.

4. Formal statement of the algorithm

The following ALGOL-like procedures specify the chain code to quadtree algorithm. The chain code is assumed to be in a linked list of records of type list. A record of type list has two fields named VALUE and NEXT which correspond respectively to the direction of a code element and a link to the record containing the next element of the code. The main procedure is named QUADTREE and is invoked with a pointer to the first element in the chain code. QUADTREE implements the first phase of the algorithm by constructing the quadtree corresponding to the border nodes. FIND_NEIGHBOR and recursive procedure TRACE_NEIGHBOR are used to locate neighboring nodes along the border. Once the tree has been obtained, recursive procedure FILLIN is invoked to fill in non-null links of GRAY nodes and to convert GRAY nodes having four BLACK nodes as sons to a BLACK node.

```
procedure QUADTREE(HEAD):
/*given a chain code in a list pointed at by HEAD, find the
  corresponding quadtree.  Each list element has two fields,
  VALUE and NEXT.  VALUE contains the value of the link in the
  code and NEXT points at the next element of the list */
begin
    list P,HEAD;
    direction OLD, NEW;
    node Q;
    P←HEAD;
    OLD←VALUE(P);
    Q←CREATENODE(NULL,NULL,CODE(BOUNDARY(OLD))); /*create the
                                                   first node in
                                                   the tree */
    P←NEXT(P)
    while not null P do  /* traverse the boundary of the region
                            and build the tree */
        begin
            NEW←VALUE(P);
            if COMPARE(OLD,NEW)=CLOCKWISE then  /*Figure 3b */
                NODETYPE(Q)←NODETYPE(Q) LOR CODE(BOUNDARY(NEW))
            else if COMPARE(OLD,NEW)=SAME then  / *Figure 3a */
                begin
                    Q←FIND_NEIGHBOR(Q,OLD);
                    NODETYPE(Q)←NODETYPE(Q) LOR CODE(BOUNDARY(OLD));
                end
```

```
            else if COMPARE(OLD,NEW)=COUNTERCLOCKWISE then /*Figure 3c* /
                begin
                    Q←FIND_NEIGHBOR(Q,OLD);
                    Q←FIND_NEIGHBOR(Q,NEW);
                    NODETYPE(Q)←NODETYPE(Q) LOR CODE(BOUNDARY(NEW));
                end
            else ERROR ("wrong direction");
            OLD←NEW;
            P←NEXT(P);
        end;
    while not null FATHER(Q) do Q←FATHER(Q); /*find the root of the
                                               tree*/
    FILLIN(Q);  /*fill in NULL links of GRAY nodes and merge
                  if possible*/
end;
```

```
node procedure FIND_NEIGHBOR(Q,S)
/* given node Q, return node P which touches side S of node Q */
begin
    node P,Q;
    direction S;
    P←TRACE_NEIGHBOR(Q,S);
    if GRAY(NODETYPE(P)) then NODETYPE(P)←NOBORDERBLACK;
        /* if this is the first visit to the node, then
           initialize its border code */
    return(P);
end;

node procedure TRACE_NEIGBHOR(Q,S);
/* given node Q, return node P which touches side S of node Q.
   This is done by finding a common ancestor of the two nodes
   and creating one if it does not exist */
begin
    node P,Q;
    direction S;
    if null SONTYPE(Q) then   /* common ancestor does not exist */
        begin
            P←CREATENODE(NULL,NULL,GRAY);  /* create a common ancestor */
            SON(P,QUAD(BOUNDARY(S),OPDIR(S)))←Q;
            FATHER(Q)←P;
            SONTYPE(Q)←QUAD(BOUNDARY(S),OPDIR(S));
        end
```

```
    else if ADJ(S,SONTYPE(Q)) then P←TRACE_NEIGHBOR(FATHER(Q),S)
    else P←FATHER(Q);
        /* trace a path from the common ancestor to the adjacent
           node creating nonterminal nodes as necessary */
    return (if null SON(P,REFLECT(S,SONTYPE(Q))) then
                CREATENODE(P,REFLECT(S,SONTYPE(Q)),GRAY)
            else SON(P,REFLECT(S,SONTYPE(Q))));
end;

procedure FILLIN(ROOT);
/* process the quadtree rooted at ROOT by filling in all NULL
   links of GRAY nodes.  Also change every GRAY node having four
   BLACK sons to a BLACK node */
begin
    node ROOT;
    quadrant ABSENT, I;
    integer B;
    if BLACK (ROOT) then return;  /* a terminal node */
    ABSENT←NULL;
    for I in {NW,NE,SE,SW} do  /*recursively process the sons of
                                 ROOT in clockwise order */
        begin
            if null SON(ROOT,I) then ABSENT←I  /* a NULL link of
                                                  a GRAY node */
            else FILLIN(SON(ROOT,I));
        end;
```

```
                for I in {NW,NE,SE,SW} do
                    DELETE(SON(ROOT,I)); /* link field is set to NULL */
            end;
end;

node procedure CREATENODE(ROOT,S,T);
/* create a node P with border code T which is the S son of node
   ROOT and return P */
begin
    node P,ROOT;
    quadrant I,S;
    integer T;
    P←GETNODE();
    if ROOT then SON(ROOT,S)←P; /* created node has a father */
    SONTYPE(P)←S;
    FATHER(P)←ROOT;
    NODETYPE(P)←T;
    for I in {NW,NE,SE,SW} do SON(P,I)←NULL;
    return (P);
end;
```

```
if not null ABSENT then
    /* ROOT has at least one GRAY node with a NULL link.
       Determine if its NULL link corresponds to a BLACK or
       WHITE node and set all of its NULL sons to the same color */
    begin
        while not (null SON(ROOT,ABSENT)
                   and SON(ROOT,CQUAD(ABSENT))) do
            ABSENT←CQUAD(ABSENT) /*find a non-NULL son adjacent
                                   to a NULL son */
        B←if ADJACENT_INTERIOR(SON(ROOT,CQUAD(ABSENT)),ABSENT,
                               CCQUAD(ABSENT),COMMONBOUNDARY
                               (ABSENT,CQUAD(ABSENT)))
              then NOBORDERBLACK
            else WHITE,
        for I in {NW,NE,SE,SW} do
            begin
                if null SON(ROOT,I) then CREATENODE(ROOT,I,B)·
            end;
    end;
if BLACK(SON(ROOT,NW)) and BLACK(SON(ROOT,NE))  and BLACK(SON
    (ROOT,SE)) and BLACK(SON(ROOT,SW)) then
    /* all four sons of node ROOT are BLACK.  Change ROOT from
       GRAY to BLACK, update the border code, and delete
       ROOT's sons */
        begin
            NODETYPE(ROOT)←Ø;
            for I in {NW,NE,SE,SW} do
                NODETYPE(ROOT)← NODETYPE(ROOT)LOR
                                (NODETYPE(SON(ROOT,I)) LAND
                                 NODETYPE(SON(ROOT,CQUAD(I))));
```

```
boolean procedure ADJACENT_INTERIOR(ROOT,Q1,Q2,T);
/* determine if the BLACK descendants in quadrants Q1 and Q2 of
   ROOT are BLACK and are not on the T border of ROOT */
begin
    node ROOT;
    quadrant Q1,Q2;
    boundary T;
    if NODETYPE(ROOT)=GRAY then
        ADJACENT_INTERIOR(SON(ROOT,Q1),Q1,Q2,T) and
        ADJACENT_INTERIOR(SON(ROOT,Q2),Q1,Q2,T)
    else BLACK(ROOT) and (CODE(T)land NODETYPE(ROOT)=Ø);
end;
```

5. Analysis

The running time of the quadtree construction algorithm is determined by the time necessary to execute its two phases. Phase one depends on the speed of procedure FIND_NEIGHBOR which constructs the tree by visiting all of the nodes which are either adjacent to or abut the border. Let LEN be the length of the chain code in terms of the number of unit vectors. This is also the perimeter of the region represented by the chain code. The number of times that FIND_NEIGHBOR is invoked is obtained by the following theorem.

Theorem: FIND_NEIGHBOR is invoked LEN-4 times

Proof: Clearly, for any closed connected region, a clockwise chain code requires a minimum of four clockwise turns. Any counterclockwise turn requires an additional clockwise turn in excess of the minimum. To see this, consider a code proceeding in the easterly direction as in Figure 9a. A counterclockwise turn, as in Figure 9b, must eventually be followed by a clockwise turn (i.e., north to east) to resume the path in the easterly direction as in Figures 9c and 9d. It should be borne in mind that every closed connected non-empty region requires a chain code with at least one link in the east, south, west, and north directions. Moreover, turns in opposite directions are prohibited.

Now, recalling Figure 3, we see that FIND_NEIGHBOR is invoked once for a link whose direction is unchanged from that of the previous link, not at all for clockwise turns, and twice for

counterclockwise turns. Since the number of counterclockwise turns is 4 less than the number of clockwise turns, we have the result that FIND_NEIGHBOR is invoked LEN-4 times. Q.E.D.

Each call to FIND_NEIGHBOR for node P and direction S requires time equal to two times the path length, termed the distance, from P to the ancestor it shares with its neighbor along direction S. In the worst case, this will require ascending from a terminal node to the root of the tree and back. The average time is obtained with the aid of the following theorem:

Theorem: The average time required to find a common ancestor is 2.

Proof: Assume that the space has been partitioned into a 2^n by 2^n array. Given a node P and a direction S, there are 2^n-1 possible positions for node P and a neighbor in direction S. Of these 2^n-1 neighbor pairs, 2^0 have their nearest common ancestor at level n, 2^1 at level n-1,...,2^i at level n-i, and 2^{n-1} at level 1. Assuming that node P is equally likely to occur at any of the 2^n-1 positions, then the average time required to find a common ancestor is as follows:

$$\frac{1}{2^n-1} \sum_{i=1}^{n} i \cdot 2^{n-1} \approx \frac{1}{2^n} \sum_{i=1}^{n} i \cdot 2^{n-i} = \sum_{i=1}^{n} \frac{i}{2^i}$$

$$\sum_{i=1}^{n} i/2^i = \frac{1}{2} \sum_{i=0}^{n-1} (i+1)/2^i$$

$$= \frac{1}{2} \sum_{i=0}^{n-1} i/2^i + \frac{1}{2} \sum_{i=0}^{n-1} i/2^i$$

$$= \frac{1}{2} \sum_{i=1}^{n-1} i/2^i + \frac{1}{2} \frac{1-\frac{1}{2^n}}{1-\frac{1}{2}}$$

$$= \frac{1}{2} \sum_{i=1}^{n} i/2^i - \frac{n}{2^n} + 1 - \frac{1}{2^n}$$

$$\therefore \quad \sum_{i=1}^{n} i/2^i = 2 - \frac{n+1}{2^{n-1}}$$

$$\simeq 2$$

Q.E.D.

Using the above result yields that on the average for each call to FIND_NEIGHBOR, 4 nodes are visited. Since FIND_NEIGHBOR is invoked LEN-4 times, we have that on the average, phase one of the algorithm takes time proportional to (LEN-4)*4.

Phase two of the algorithm depends on the speed of procedure FILLIN. It has four parts, each of which is analyzed separately below. An exact analysis depends on the relative positions of the various nodes. However, based on empirical observations of regions we make a number of assumptions and obtain a reasonable estimate for the running time of FILLIN.

Assume that the space has been partitioned into a 2^n by 2^n array. Let the quadtree obtained as a result of phase one contain m nodes at level 0 (i.e., in BLACK leaf nodes). Assume further that at each level (other than $\emptyset$) of the tree, each GRAY node has two sons. This is a reasonable assumption when the region does not contain too many jagged edges. Therefore, there are $\frac{m}{2^i}$ GRAY

nodes at level i where i ranges between 1 and n. Thus the tree contains approximately 2m nodes. These assumptions are based on the observation that as the tree is built the nodes that touch or abut the border are adjacent to each other. Most often successive pairs of these nodes are sons of the same node, say Q, and if the code does not turn, the remaining two sons of Q are left NULL to be colored later as BLACK or WHITE by FILLIN. The same reasoning applies to the GRAY nodes at level 1 through n. For example, consider Figure 8 which shows the result of the application of phase one of the algorithm to Figure 7. There are 43 BLACK nodes at level $\emptyset$, 21 GRAY nodes at level 1, 10 GRAY nodes at level 2, 4 GRAY nodes at level 3, and 1 GRAY node at level 4. Note that our assumptions are less accurate at the level of the root of the tree.

The first part of procedure FILLIN visits each node in the quadtree once in order to determine if it is a BLACK node (i.e., if it is a terminal node). Using our earlier assumptions with respect to the size of the tree results in 2^{m-1} nodes being visited.

The second part of FILLIN visits non-terminal nodes to determine a pair of adjacent sons one of which is NULL while the other is not. Using our earlier assumptions with respect to tree size results in m-1 nodes being visited.

The third part of FILLIN attempts to fill in the NULL links. Recall, from the description of phase two, our observation that only one NULL link of any non-terminal node, say P, must be so examined (i.e., the remaining NULL links will have the same color). This is done by procedure ADJACENT_INTERIOR which traverses a subtree of P that is adjacent to the NULL link. Note that only sons that are adjacent to the border shared by P and the region corresponding to the NULL link need to be examined. In other words, the maximum number of nodes visited when P is at level i is the size of a complete binary tree of height i-1, i.e., 2^i-1. Assuming the worst case that ADJACENT_INTERIOR is applied to one of the two sons of each node at level i, we have $\frac{m}{2^i}$ sons being visited. Therefore, the number of visits made by ADJACENT_INTERIOR is:

$$\sum_{i=1}^{n} (2^i-1)\frac{m}{2^i} = m \sum_{i=1}^{n} (1-\frac{1}{2^i})$$

$$= mn-m \sum_{i=1}^{n} \frac{1}{2^i}$$

$$= mn-m \left(\sum_{i=0}^{n} \frac{1}{2^i} - 1 \right)$$

$$= mn-m \left(\frac{1-\frac{1}{2^{n+1}}}{1-\frac{1}{2}} - 1 \right)$$

$$= mn-m \left(2-\frac{1}{2^n} - 1 \right)$$

$$= mn - m\left(1 - \frac{1}{2^n}\right)$$

$$\approx mn - m$$

$$= m(n-1)$$

The fourth part of FILLIN attempts to convert each GRAY node having four BLACK sons to a BLACK node. Our assumptions stipulate that at level i ($1 \le i \le n$) there exist $\frac{m}{2^i}$ GRAY nodes. Therefore, at level i, this part will visit $\frac{m}{2^i} * 4$ nodes. Summing over all i yields:

$$\sum_{i=1}^{n} \frac{m}{2^i} * 4 = 4m \sum_{i=1}^{n} \frac{1}{2^i}$$

$$= 4m\left(\sum_{i=0}^{n} \frac{1}{2^i} - 1\right)$$

$$= 4m\left(\frac{1 - \frac{1}{2^{n+1}}}{1 - \frac{1}{2}} - 1\right)$$

$$= 4m\left(2 - \frac{1}{2^n} - 1\right)$$

$$\approx 4m$$

Therefore, procedure FILLIN makes approximately (n+6)m visits to the nodes in the tree. Note that $m \le LEN$ and can be approximated by it. Thus from the analysis of phase one and phase two we have the following theorem:

Theorem: The quadtree is obtained in average case time proportional to the product of the perimeter and the log of the diameter

of the region.

Proof: Phase one and phase two have an average case execution time proportional to (n+10)*LEN as derived above. Q.E.D.

6. Concluding remarks

An algorithm has been presented for converting a chain code description of a simply-connected region's boundary into a quadtree representation of the region. The algorithm's running time is proportional to the product of the region's perimeter and the log of its diameter when certain assumptions are made about the shape of the region.

This algorithm, coupled with the algorithm in [DRS] for constructing a chain code representation for a quadtree, enables the use of the quadtree as a convenient data structure for describing regions. In particular, operations can be performed using the particular representation which results in the greatest efficiency.

As a final observation we note that the trees that we have constructed do not necessarily have a minimum number of nodes. An interesting question for future research is the degree of deviation of our trees from the trees with a minimal number of nodes. Techniques similar to those of [Hunter] could be used to transform our trees into minimal trees.

References

[DRS] C. R. Dyer, A. Rosenfeld, and H. Samet, Region representation: boundary codes from quadtrees. TR-732, Computer Science Center and Computer Science Department, University of Maryland, College Park, February 1979.

[FB] R. A. Finkel and J. L. Bentley, Quadtrees: a data structure for retrieval on composite keys, Acta Informatica 4, 1974, 1-9.

[Freeman] H. Freeman, Computer processing of line-drawing images, Computing Surveys 6, 1974, 57-97.

[Hunter] G. M. Hunter, Efficient computation and data structures for graphics, Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, New Jersey, 1978.

[KD] A. Klinger and C. R. Dyer, Experiments in picture representation using regular decomposition, Computer Graphics and Image Processing 5, 1976, 68-105.

[Naur] P. Naur (Ed.), Revised report on the algorithmic language ALGOL 60, Communications of the ACM 3, 1960, 299-314.

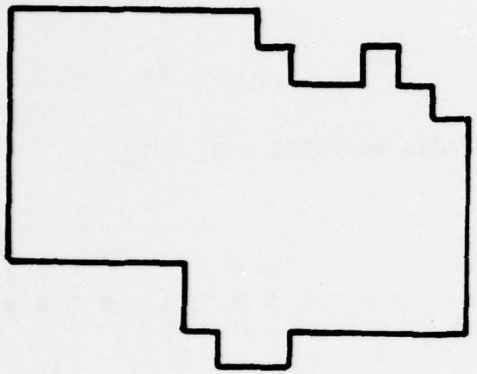


Fig. 1a. Sample region.

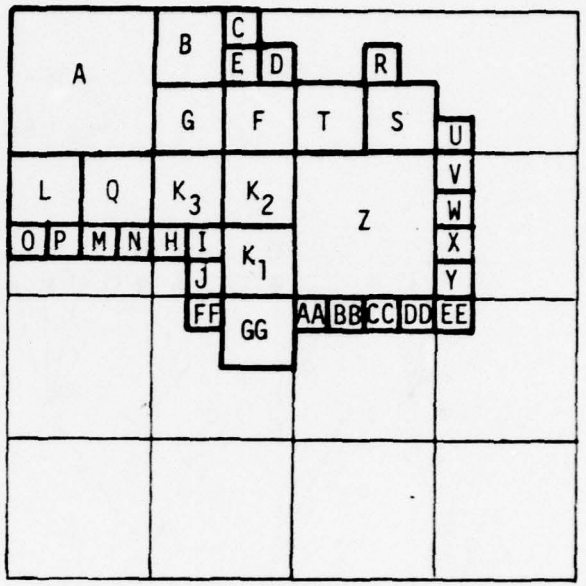


Fig. 1b. Block decomposition of the region in Fig. 1a.

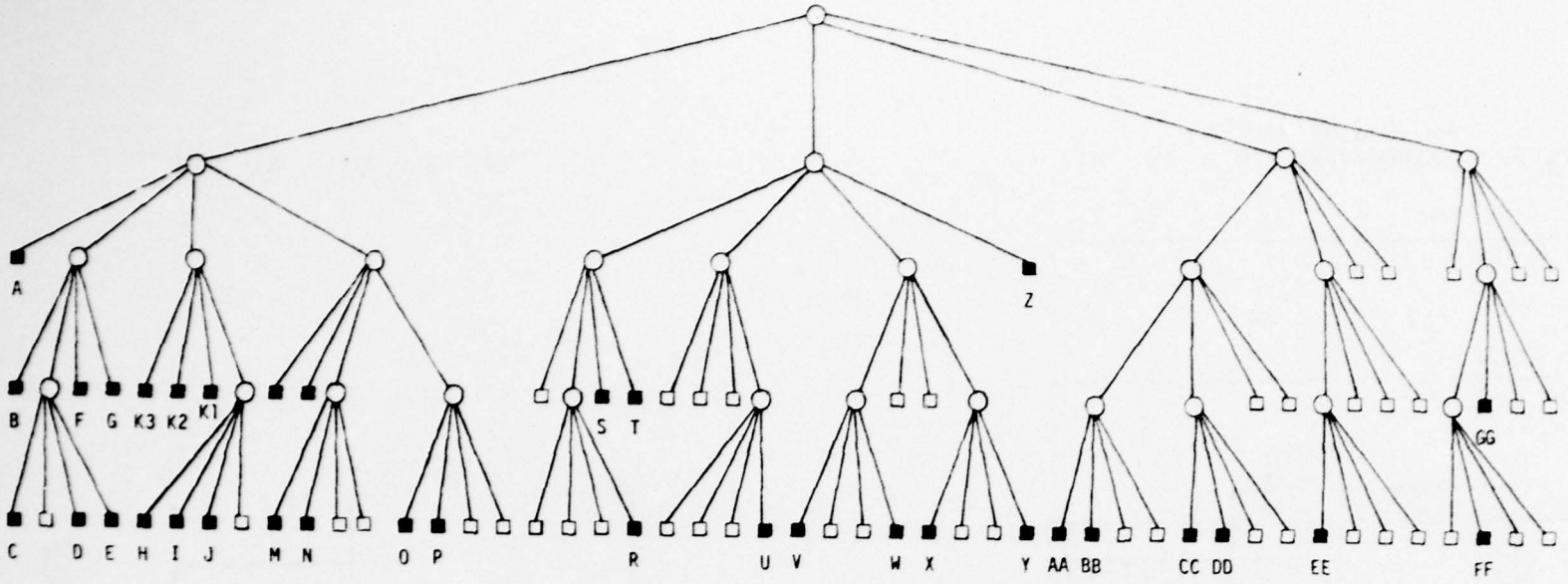
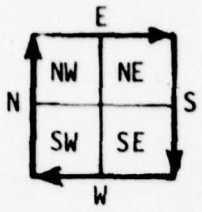


Fig. 1c - Quadtree representation of the blocks in Fig. 1b.



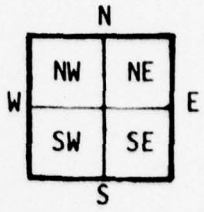


Fig. 2a - Relationship between a block's four quadrants and its chain code representation.

Fig. 2b - Relationship between a block's four quadrants and its boundary.

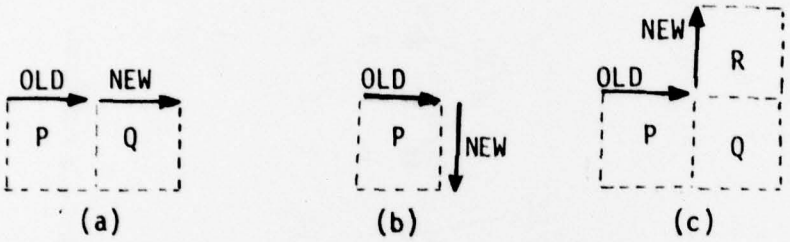


Fig. 3 - Examples of the actions to be taken when the code maintains its direction (a), turns clockwise (b), and turns counterclockwise.

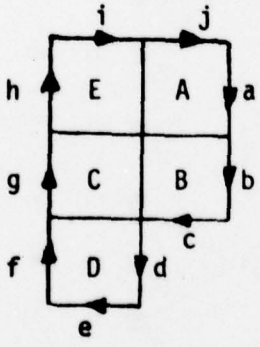


Fig. 4 - Sample region and its chain code.

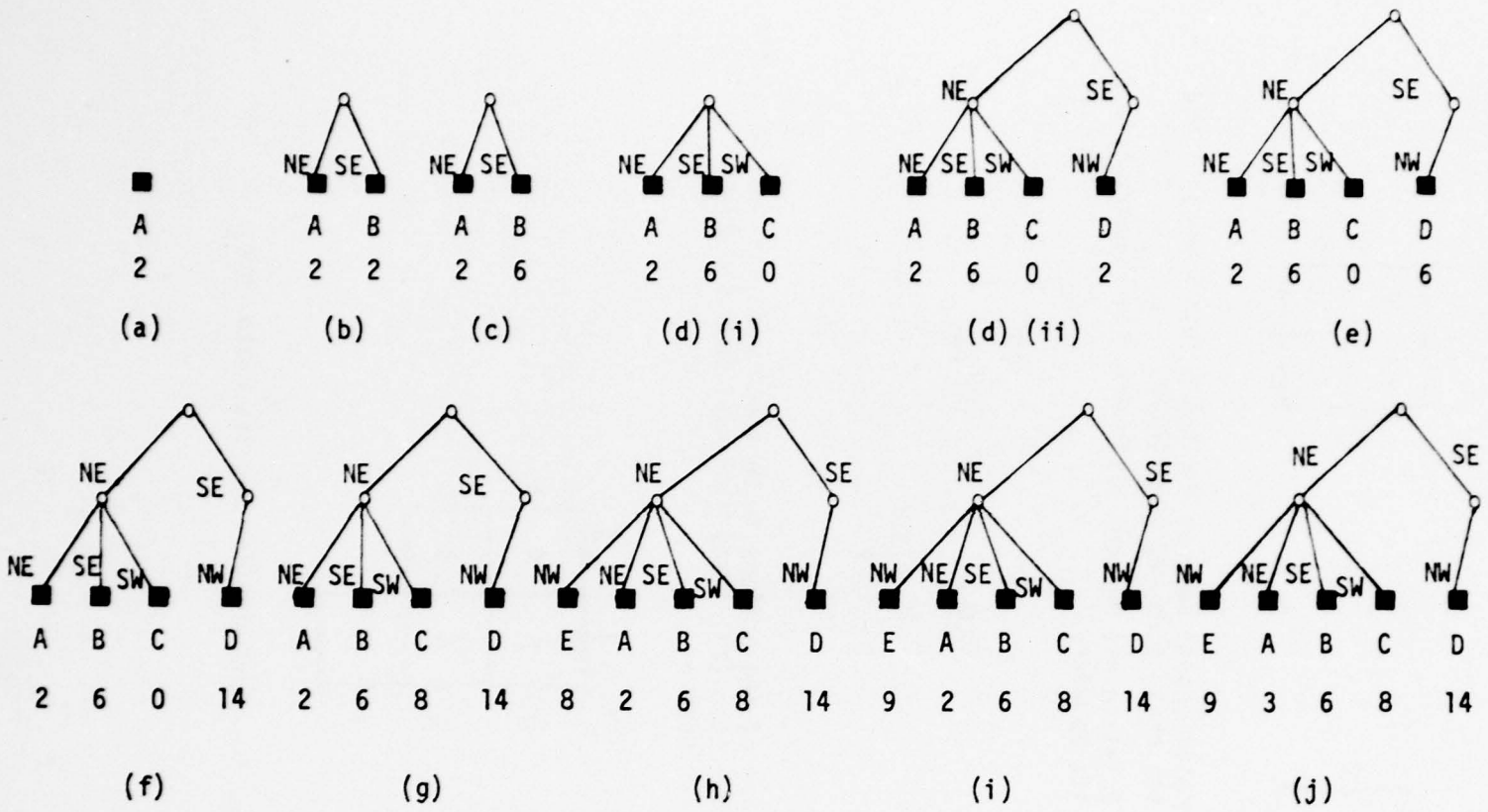


Fig. 5 - Intermediate trees in the process of obtaining a quadtree correspondong to Fig. 4.

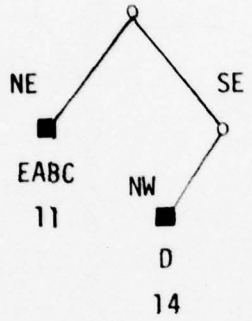


Fig. 6 - Result of the application of phase two of the algorithm to Fig. 5.

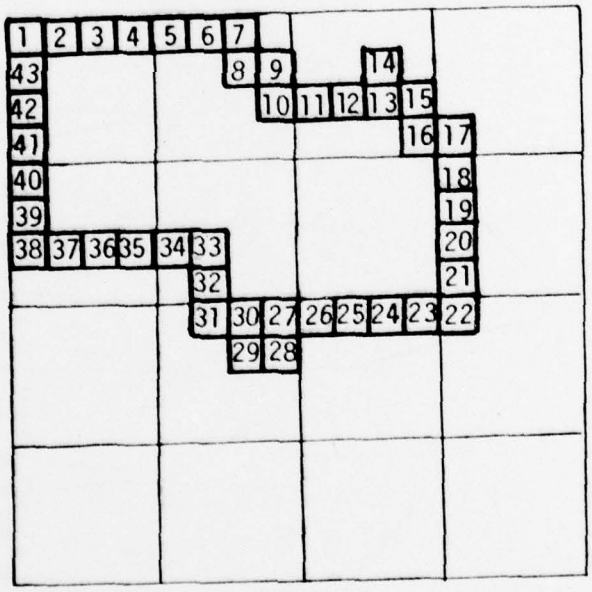


Fig. 7. Block decomposition of the region in Fig. 1a after application of phase one of the chain code to quad tree algorithm.

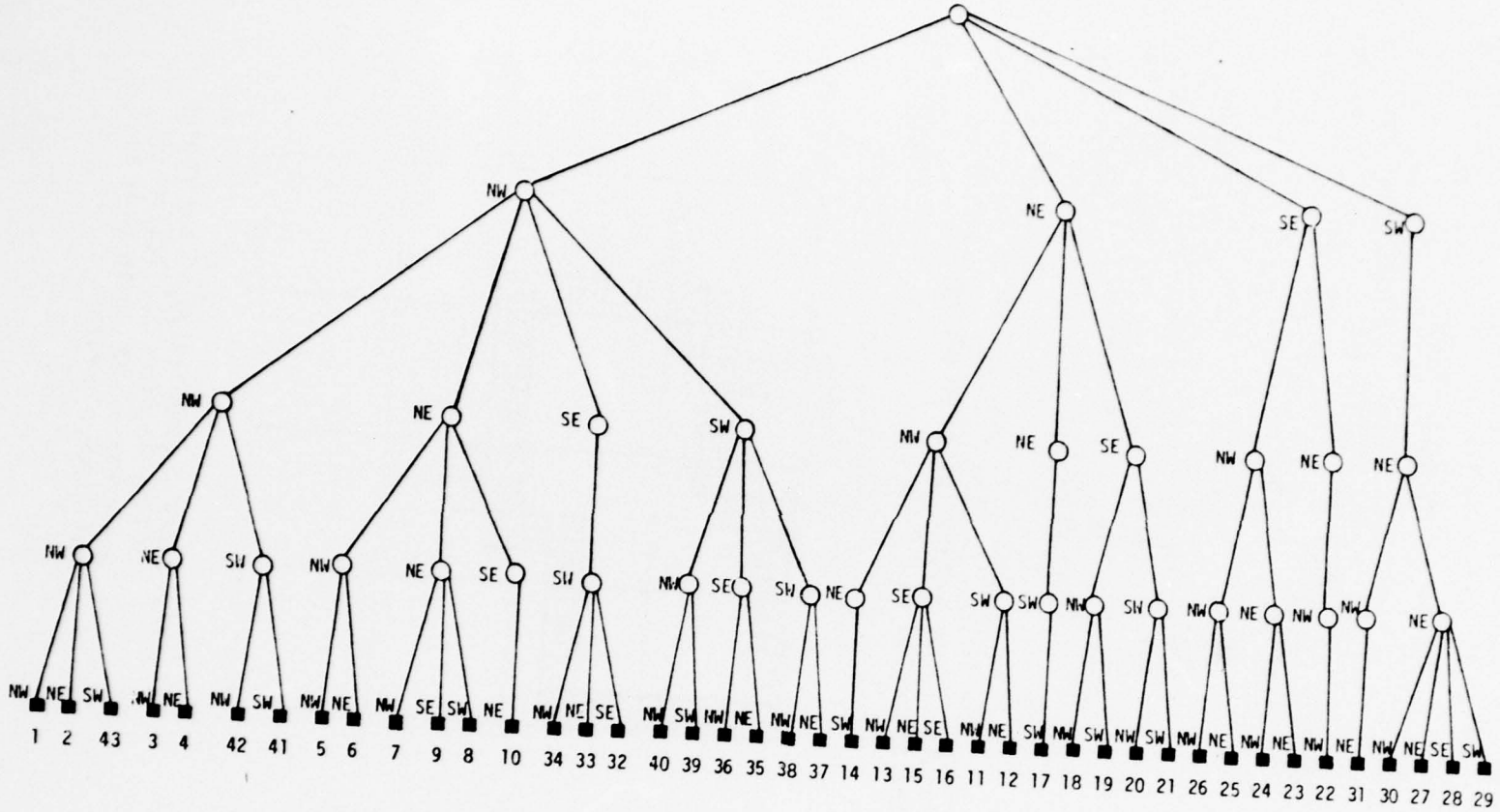


Fig. 8 - Quadtree corresponding to the region in Fig. 1a after application of phase one of the chain code to quadtree algorithm.

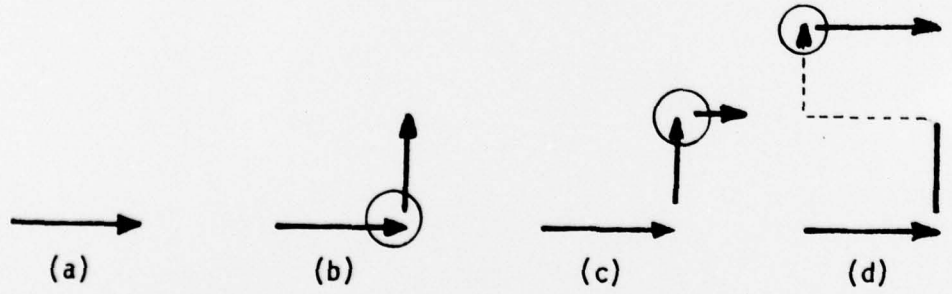


Fig. 9 - Sequence of turns illustrating the need for one clockwise turn for each counterclockwise turn.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) REGION REPRESENTATION: QUADTREES FROM BOUNDARY CODES		5. TYPE OF REPORT & PERIOD COVERED Technical 6. PERFORMING ORG. REPORT NUMBER TR-741
7. AUTHOR(s) Hanan Samet		8. CONTRACT OR GRANT NUMBER(s) DAAG-53-76C-0138
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Center University of Maryland College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Night Vision Laboratory Fort Belvoir, VA 22060		12. REPORT DATE March 1979 13. NUMBER OF PAGES 39
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image processing
Pattern recognition
Cartography
Region representation
Quadtrees
Boundary codes

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An algorithm is presented for constructing a quadtree for a region given its boundary in the form of a chain code. The algorithm makes use of some geometrical properties of the region to enable the detection of the maximal size blocks of the region without actually visiting all of the subblocks of the maximal size block. Analysis of the algorithm reveals that its execution time is proportional to the product of the perimeter and the log of the diameter of the region.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)